Parallelizing the Spectral Transform Method: A Comparison of Alternative Parallel Algorithms*

Ian Foster† Patrick H. Worley‡

Abstract

The spectral transform method is a standard numerical technique for solving partial differential equations on the sphere and is widely used in global climate modeling. In this paper, we outline different approaches to parallelizing the method and describe experiments that we are conducting to evaluate the efficiency of these approaches on parallel computers. The experiments are conducted using a testbed code that solves the nonlinear shallow water equations on a sphere, but are designed to permit evaluation in the context of a global model. They allow us to evaluate the relative merits of the approaches as a function of problem size and number of processors. The results of this study are guiding ongoing work on PCCM2, a parallel implementation of the Community Climate Model developed at the National Center for Atmospheric Research.

1 Introduction

Parallel algorithms for computing the spectral transform method used in climate models can be divided into two general classes. *Transform algorithms* employ a single decomposition of the principal data structures and use parallel Legendre transforms (LTs) and fast Fourier transforms (FFTs) to communicate data among the different partitions. An algorithm of this type is being used to construct a parallel implementation [4] of the National Center for Atmospheric Research's Community Climate Model (CCM2) [13]. In contrast, *transpose algorithms* use different data decompositions at different stages of the computation, permitting the LTs and FFTs to proceed without communication. Parallel matrix transpose operations are used to move between the different decompositions. An algorithm of this type has been used at the European Center for Medium-Range Weather Forecasts to execute their production spectral model on an eight-processor Cray Y/MP [2]. Hybrid algorithms, in which for example a transpose FFT is combined with a parallel LT, are also possible.

A comprehensive comparison of these two approaches to the parallel implementation of the spectral transform method has not previously been attempted. In this paper, we describe analytic and empirical studies intended to provide a detailed understanding of the relative efficiency of the two approaches as a function of both problem size and machine characteristics. This work focuses on mesh-connected multicomputers with cut-through routing, such as the Intel Delta and Paragon computers. However, it is easily extended to other parallel computer architectures. An important goal of this work is to evaluate potential refinements to the parallel implementation of CCM2 currently under development

*This work was supported by the Atmospheric and Climate Research Division of the Office of Energy Research, U.S. Department of Energy.

†Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439.

‡Oak Ridge National Laboratory, P.O. Box 2008, Bldg. 6012, Oak Ridge, TN 37831-6367.

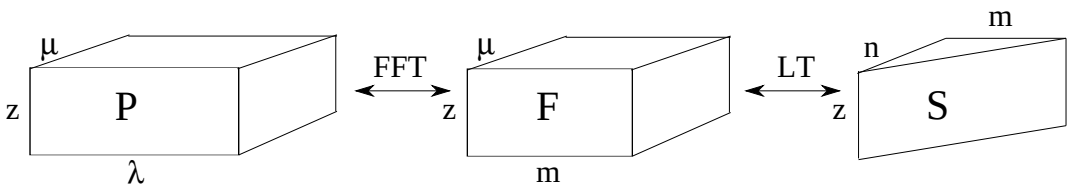

FIG. 1. *Principal Data Structures in Spectral Transform*

at Argonne and Oak Ridge National Laboratories as part of the Department of Energy CHAMMP initiative [3]. Hence, we also discuss other issues that must be considered when applying the results of this study to parallel implementations of climate models.

2 Spectral Transform Method

In the spectral transform method, state variables are transformed at each timestep between the physical domain, where most of the physical forces are calculated, and the spectral domain, where the terms of the differential equation are evaluated. In the codes that we consider, the time update and all coupling between vertical layers are calculated in physical space. The spectral representation of a state variable ξ on a given vertical layer above the surface of a sphere is defined by an approximation to the variable by a truncated series of spherical harmonic functions,

$$\xi(\lambda,\mu) = \sum_{m=-M}^{M} \sum_{n=|m|}^{N(m)} \xi_n^m P_n^m(\mu) e^{im\lambda},$$

where $\mu = \sin\theta$, θ is latitude, λ is longitude, and $P_n^m(\mu)$ is the associated Legendre function. In the physical domain, state variables are approximated on an $I \times J$ longitude-latitude grid. Transforming from physical coordinates to spectral coordinates involves performing an FFT for each line of constant latitude, followed by integration over latitude using Gaussian quadrature to obtain the spectral coefficients,

$$\xi_n^m = \sum_{j=0}^{J-1} \xi^m(\mu_j)\, P_n^m(\mu_j)\, w_j,$$

where ξ^m is the mth Fourier coefficient, and w_j is the Gaussian quadrature weight corresponding to Gaussian latitude μ_j. The point values are recovered from the spectral coefficients by computing

$$\xi^m(\mu) = \sum_{n=|m|}^{N(m)} \xi_n^m P_n^m(\mu)$$

for each m, followed by FFTs to calculate $\xi(\lambda,\mu)$.

When the spectral transform method is applied in a global climate model, the principal data structures are as shown in Figure 1. In this figure we assume that $N(m) = M$ and P denotes physical space, F Fourier space, S spectral space, and z the number of vertical layers in the model.

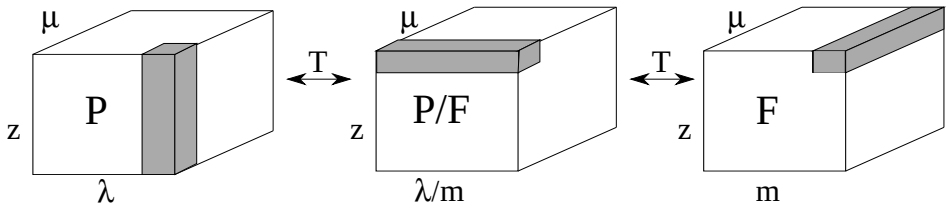

FIG. 2. *Parallel Transpose Algorithm*

3 Parallel Transform Algorithms

Parallel transform algorithms decompose the principal data structures by latitude and longitude. Parallel implementations of the FFT and LT are then required. The parallel Fourier transform algorithm used in this work [14] extends the conventional power-of-two parallel FFT algorithm by performing two block transforms concurrently to overlap computation with communication.

We consider two alternative algorithms for the parallel LT [4]. The first, `ring-pipeline`, uses a pipeline algorithm over a logical ring of processors to overlap communication with computation during the LT. The second, `gadd`, computes as much as possible locally before calling a fast global vector sum routine to finish the computation. `ring-pipeline` partitions the spectral domain equally while subsets of processors "share" the same spectral coefficients in `gadd`. In consequence, some computation in the spectral domain is calculated redundantly in `gadd`, but the calculation of $\{\xi^m(\mu)\}$ from $\{\xi_n^m\}$ requires no interprocessor communication because all necessary information is available locally.

4 Parallel Transpose Algorithms

Transpose algorithms for the spectral transform employ matrix transpose operations to move between alternative decompositions of basic data structures, so that the FFTs and LTs do not involve communication. This approach is illustrated in Figure 2, in which the data allocated to a single processor at different stages of the computation is shaded. Initially, the physical space variables are decomposed by latitude and longitude so that computations involving vertical dependencies can proceed without communication. A transpose is then performed to obtain a decomposition by latitude and vertical layer, allowing the FFTs to proceed without communication. A second transpose provides a decomposition by longitude and vertical layer, allowing the LT to proceed without communication. Reverse transposes are then performed to return to the original decomposition.

On $p \times q$ processors, the first transpose involves q independent p-processor transposes, while the second requires p independent q-processor transposes.

5 Performance Models

We have developed detailed performance models for the parallel transpose and transform algorithms. Because of space constraints, we are able to sketch only some aspects of these models here.

In previous work, we have shown that the communication costs for a parallel Fourier and ring-pipeline Legendre transform of r data elements can be approximated as follows [6]:

$$O_{fft} = t_s \log p + t_h 2(\sqrt{p} - 1) + t_w 2r \frac{\sqrt{p} - 1}{p} \tag{1}$$

$$O_{rp} = (p-1)\left(t_s + t_h + t_w \frac{r}{p}\right). \quad (2)$$

In these expressions, p is the number of processors and the parameters t_s, t_h, and t_w represent message startup costs, per-hop costs, and per-word costs in a mesh-connected distributed-memory parallel computer with cut-through routing. The FFT performance model assumes that competition for bandwidth in a two-dimensional mesh computer increases the transfer costs from $\log p$ to $\sqrt{p}$.

A matrix of r data elements partitioned among p processors can be transposed in $p-1$ communication steps per processor, each involving r/p^2 data elements. In a mesh architecture, the average distance traveled is about $\sqrt{p}$. Hence, in the absence of competition for bandwidth we can approximate the total communication cost by

$$(p-1)\left(t_s + t_h\sqrt{p} + t_w \frac{r}{p^2}\right).$$

Competition for bandwidth must be taken into account. In $p-1$ communication steps, a total of approximately $p(p-1)\sqrt{p}$ hops must be traversed (as p processors are communicating). As there are only $4p$ communication links available at each step, we have an average of $\sqrt{p}/4$ messages per link. Scaling the number of words to be communicated by this amount, we arrive at an approximate communication cost for a single transpose of

$$(p-1)\left(t_s + t_h\sqrt{p} + t_w \frac{r}{4p^{1.5}}\right). \quad (3)$$

This analysis provides a lower bound on the cost of performing a transpose on a 2-D mesh. This lower bound can be achieved only if we can define a communication schedule that allows p processors to perform a transpose in $p^{1.5}/4$ phases without contention. Scott provides a constructive proof that such a schedule exists in an $p \times p$ mesh with p a multiple of four [11], demonstrating that this lower bound is achievable.

An alternative transpose algorithm performs the transpose in $\log p$ steps [9]. This performs fewer communications but transfers more data and must perform additional copying. The algorithm can be expected to be faster in situations where message startup costs dominate: for example, on clustered workstations or on small problems.

6 Empirical Studies

Although analytic models can provide interesting insights into performance issues, empirical studies are required to calibrate and validate models. To permit a fair comparison of the suitability of the transpose and transform algorithms for general circulation models such as CCM2, we have incorporated the various algorithms in a single testbed code with a structure similar to that of CCM2.

6.1 STSWM: An Algorithm Testbed

Our testbed code is based on the sequential FORTRAN code STSWM developed by J. Hack and R. Jacob of NCAR for numerical studies of the shallow water equations [8]. STSWM uses the spectral transform method to solve the nonlinear shallow water equations on a sphere. The nonlinear shallow water equations constitute a simplified atmospheric-like fluid prediction model that exhibits many of the features of more complete models. They are frequently used to investigate and compare numerical methods [1]. An important

characteristic from our point of view is that STSWM's data structures and implementation of the spectral transform algorithm are based directly on equivalent structures and algorithms in CCM2.

The parallel testbed code differs from STSWM in one major respect: vertical layers have been added to permit a fair evaluation of the transpose algorithms. This is necessary because in a one-layer model, a parallel transpose algorithm reduces to a one-dimensional decomposition of each domain (see Figure 2) and hence can utilize only a small number of processors. The addition of vertical layers also has the advantage of modeling more accurately the granularity of the dynamics computation in CCM2. In all other respects we have changed STSWM as little as possible. In particular, we have not changed loop and array index ordering. Although such changes could probably improve performance of some algorithms, our goal was to have a code as similar to CCM2 as possible.

The testbed code is structured so that a variety of different parallel algorithms can be selected by runtime parameters and/or compile-time switches. The FFT can be implemented by using a parallel transform, an $\mathcal{O}(p)$ transpose, or an $\mathcal{O}(\log p)$ transpose. The LT can be implemented using a `ring-pipeline` transform, a `gadd` transform, an $\mathcal{O}(p)$ transpose, or an $\mathcal{O}(\log p)$ transpose. Hence, there are a total of twelve different algorithms to be compared.

In some of the algorithms, we are interested in exploring several different mappings of data to processors. Hence, we specify the mapping of "logical processors" to "physical processors" in a separate routine. This use of information-hiding techniques [5] makes it straightforward to explore alternative mappings. We use a similar technique to simplify the implementation of the three-dimensional transpose: a vector of processor numbers passed as an argument to a two-dimensional transpose code specifies the processors involved in a particular transpose operation.

6.2 Experimental Method

Computational experiments are being performed to compare the twelve algorithms. These experiments are designed to identify if and when one algorithm is superior to another as a function of problem size, number of processors, and underlying machine topology. They will also provide the data required to validate and improve our performance models. This will allow us to make meaningful performance predictions for future parallel computers (e.g., the 2048-processor Intel Paragon to be installed at Oak Ridge National Laboratory) and for related codes with somewhat different computational characteristics (e.g., the NCAR Community Climate Model).

The testbed code is implemented using PICL [7], a portable message-passing library that incorporates instrumentation. This provides portability between Intel and NCUBE multiprocessors and allows the collection of data for performance studies. In addition, we are using the PICL emulation mode in W. Gropp's portable Chameleon package, both to develop the testbed programs on a workstation and to execute the parallel code on machines to which PICL has not yet been ported.

6.3 Results

We report results obtained in two preliminary experiments intended to validate aspects of the parallel transform and transpose performance models.

The first study explores the accuracy of the LT model, Equation 2. We fit a performance model comprising Equation 2 plus a characterization of sequential computation time to

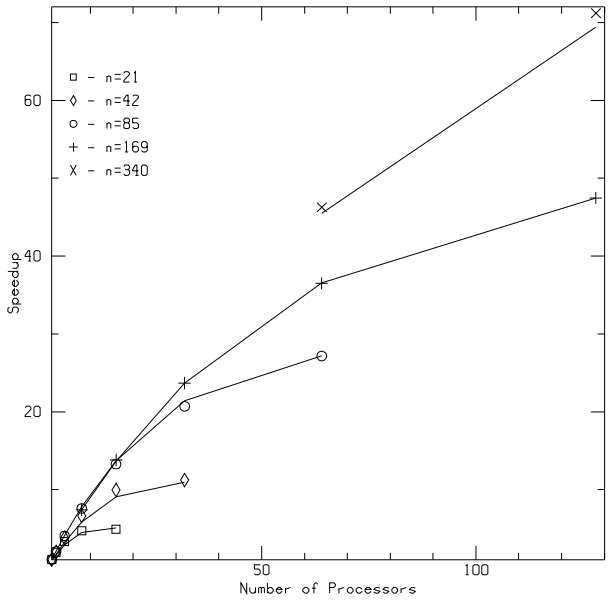

FIG. 3. *Predicted (lines) and Observed (data points) Speedups for Transform Code*

runtimes obtained using an early version of the testbed code [15]. These experiments were conducted on a 128-processor Intel iPSC/860 computer. Figure 3 shows the fit obtained using the following values for t_s, t_h, and t_w, with observed speedup indicated by points and predicted speedup by solid lines, for a variety of different problem sizes:

$$t_s + t_h = 230\ \mu\text{sec},\ t_w = 1.2\ \mu\text{sec}.$$

Because the time complexity for the floating-point computation contains terms with the same asymptotic form as the principal communication terms, it is impossible to determine an exact correspondence between the t_s, t_h, and t_w values obtained by fitting and the "best achievable" communication parameters observed on the Intel iPSC/860 by other users ($t_s = 136\mu$sec, $t_h = 2.0\mu$sec, $t_w = 1.6\mu$sec). However, the similarities in the values give us confidence that our model is accurate and the parallel implementation is efficient. The disparities may be artifacts of the fitting process or may reflect additional operations (e.g., buffer formatting and management) or optimizations (e.g., overlapping of communication and computation) not dealt with by the model.

The second performance study explores the accuracy of the parallel transpose performance model, Equation 3. We fit this model to runtimes obtained using a transpose code on the 528-processor Intel Delta computer. This code performs a transpose of a two-dimensional array using the $\mathcal{O}(p)$ algorithm. Figure 4 shows the fit obtained using the following values for t_s, t_h, and t_w, with observed speedup indicated by points and predicted speedup by solid lines, for a variety of different problem sizes:

$$t_s = 150\ \mu\text{sec},\ t_h = 0.2\ \mu\text{sec},\ t_w = 0.32\ \mu\text{sec}.$$

These results compare well with communication parameters observed in simpler testbed codes on the Delta: $t_s = 100\mu$sec, $t_h = 0.05\mu$sec., and $t_w = 0.32\mu$sec. The mismatch between predicted and observed times at high processor counts may be due to contention.

FIG. 4. *Predicted (lines) and Observed (data points) Speedups for Transpose Code*

7 Discussion

The development of a testbed code such as the parallel STSWM raises interesting design issues. Our primary motivation in developing this code is to obtain insights into the relative merits of different parallel algorithms and hence to provide guidance to developers of parallel spectral climate models. In order for the results of our experiments to be meaningful, we must take care that all algorithms are (a) optimized to more or less the same level, and (b) implemented in a way that is consistent with the data structures used in spectral climate models. Here, we discuss other issues that must be addressed when selecting algorithms for use in a parallel climate model.

A potentially significant advantage of the parallel transpose algorithm is that it is able to use sequential FFT and LT libraries optimized (perhaps using assembly code) for a particular processor. In contrast, a parallel transform algorithm must use custom parallel FFT and LTs such as those described in [15, 14], which are less likely to be assembly coded. From a scientific viewpoint, a fair comparison of the two algorithms should probably forbid use of optimized sequential libraries. However, potential users of these algorithms will certainly be interested in results obtained with optimized libraries.

The parallel FFT described in [14] can deal only with power-of-two vectors. A non-power-of-two parallel FFT has been developed by David Semeraro, based on the Bluestein algorithm [12], but is considerably less efficient. The `gadd` and $\mathcal{O}(\log p)$ transpose algorithms are also more efficient when applied to power-of-two problems. In contrast, the $\mathcal{O}(p)$ transpose algorithm can operate on matrices of any size with equal efficiency. A complete algorithm comparison should include results for both power-of-two and non-power-of-two problems.

The `ring-pipeline` LT algorithm has been shown in previous work to be somewhat more efficient than the `gadd` algorithm on the Intel iPSC/860. However, the latter algorithm is easier to implement in a parallel climate model. Hence, it is important not only to quantify the performance difference between the two algorithms but also to describe clearly what is involved in implementing each algorithm.

Load imbalances can occur in climate models as a result of variations in the cost of approximating physical processes such as radiation and convection [10]. These load imbalances can be corrected by dynamically redistributing data prior to calling the relevant routines, which operate on data stored in physical space. This redistribution appears cheaper to perform in a transpose code. Hence, a complete algorithm comparison should perhaps investigate the relative costs of redistribution.

The various algorithms are able to use differing numbers of processors. If one assumes a computational grid of size $N_{lat} \times N_{lon} \times N_{ver}$, a pure transform algorithm constructed with the parallel FFT can use at most $(N_{lon}/4) \times (N_{lat}/2)$ processors. A pure transpose algorithm can use at most $(N_{lat}/2) \times N_{ver}$ processors. A typical problem size for a parallel climate model might be $N_{lat} = 128$, $N_{lon} = 256$, and $N_{ver} = 17$; at this resolution, a pure transform code is able to use 4096 processors, while the transpose code can use only 1088.

References

[1] G. L. Browning, J. J. Hack, and P. N. Swarztrauber, *A comparison of three numerical methods for solving differential equations on the sphere*, Mon. Wea. Rev. 117, 1989, pp. 1058–1075.

[2] D. Dent, *The ECMWF model on the Cray Y-MP8*, Proc. 4th ECMWF Workshop on Use of Parallel Processors in Meteorology, ECMWF, Reading, U.K., 1990.

[3] Department of Energy, *Building an Advanced Climate Model: Progress Plan for the CHAMMP Climate Modeling Program*, DOE Tech. Report DOE/ER-0479T, U.S. Department of Energy, Washington, D.C., 1990.

[4] J. B. Drake, R. E. Flanery, I. T. Foster, J. J. Hack, J. G. Michalakes, R. L. Stevens, D. W. Walker, D. L. Williamson, and P. H. Worley, *The message passing version of the parallel community climate model*, Proc. 5th ECMWF Workshop on Parallel Processing in Meteorology, ECMWF, Reading, U.K., 1992.

[5] I. Foster, *Information hiding in parallel programs*, Preprint MCS-P290-0292, Mathematics and Computer Science Division, Argonne National Laboratory, 1992.

[6] I. Foster, W. Gropp, and R. Stevens, *The parallel scalability of the spectral transform method*, Mon. Wea. Rev., 120(5), 1992, pp. 835–850.

[7] G. A. Geist, M. T. Heath, B. W. Peyton and P. H. Worley, *PICL: A Portable Instrumented Communication Library, C Reference Manual*, Tech. Rep. ORNL/TM-11130, Oak Ridge National Laboratory, Oak Ridge, Tenn., 1990.

[8] J. J. Hack and R. Jakob, *Description of a Global Shallow Water Model Based on the Spectral Transform Method*, NCAR Technical Note TN-343+STR, NCAR, Boulder, Colo., 1992.

[9] C-T. Ho and S. L. Johnsson, *Matrix Transposition on Boolean N-Cube Configured Ensemble Architectures*, Yale report YALEU/DCS/TR-494, 1986.

[10] J. Michalakes, *Analysis of Workload and Load Balancing Issues in the NCAR Community Climate Model*, ANL/MCS-TM-144, Argonne National Laboratory, Argonne, Ill., 1991.

[11] D. Scott, *Efficient all-to-all communication patterns in hypercube and mesh architectures*, Proc. 6th Distributed Memory Computer Conf., IEEE Computer Society Press, 1991.

[12] P. N. Swarztrauber, W. L. Briggs, R. A. Sweet, V. E. Henson, and J. Otto, *Bluestein's FFT for arbitrary N on the hypercube*, Parallel Computing 17(6), 1991, pp. 607–618.

[13] D. J. Williamson, J.T. Kiehl, V. Ramanathan, R.E. Dickinson, and J.J. Hack, *Description of NCAR Community Climate Model (CCM1)*, NCAR Technical Note TN-285+STR, NCAR, Boulder, Colo., 1987.

[14] D. W. Walker, P. H. Worley, and J. B. Drake, *Parallelizing the spectral transform method - part 2*, Concurrency: Practice and Experience 4(7), 1992, pp. 509–531.

[15] P. H. Worley and J. B. Drake, *Parallelizing the spectral transform method*, Concurrency: Practice and Experience, 4(4), 1992, pp. 269–291.